Training Deep Neural Networks

Deep Learning (DSE316/616)

Vinod K Kurmi
Assistant Professor, DSE

Indian Institute of Science Education and Research Bhopal

Sep 1, 2022

Disclaimer

- Much of the material and slides for this lecture were borrowed from
  - Bernhard Schölkopf's MLSS 2017 lecture,
  - Tommi Jaakkola's 6.867 class,
  - CMP784: Deep Learning Fall 2021 Erkut Erdem Hacettepe University
  - Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class
  - Hongsheng Li's ELEG5491 class
  - Tsz-Chiu Au slides
  - Mitesh Khapra Class notes

Previous class:Bias / Variance and Model Complexity

$$E_P\left[\left(y - h(x)\right)^2\right]$$

$$= E_P[h(x)^2 - 2yh(x) + y^2]$$

$$= E_P[h(x)^2] + E_P[y^2] - 2E_P[y]E_P[h(x)]$$

$$= E_P\left[\left(h(x) - \overline{h(x)}\right)^2\right] + \overline{h(x)}^2 + E_P\left[\left(y - f(x)\right)^2\right] + f(x)^2 - 2f(x)\overline{h(x)}$$

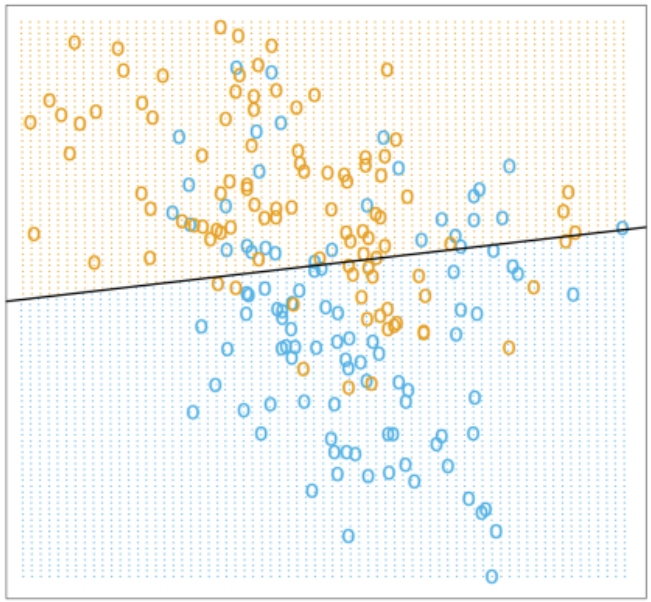
$$= E_P\left[\left(h(x) - \overline{h(x)}\right)^2\right] + \left(\overline{h(x)} - f(x)\right)^2 + E_P\left[\left(y - f(x)\right)^2\right]$$

Variance | Bias^2 | Noise

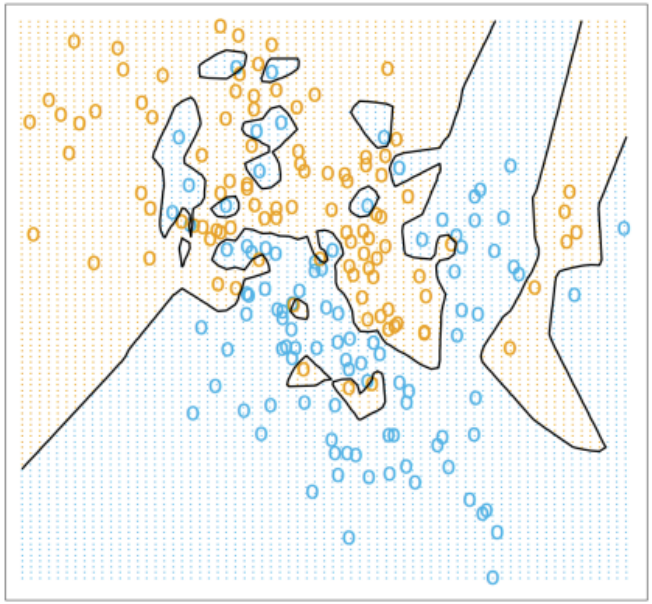
Previous class:Bias-Variance Decomposition

Simple models have high bias, but low variance.

Linear Regression of 0/1 Response



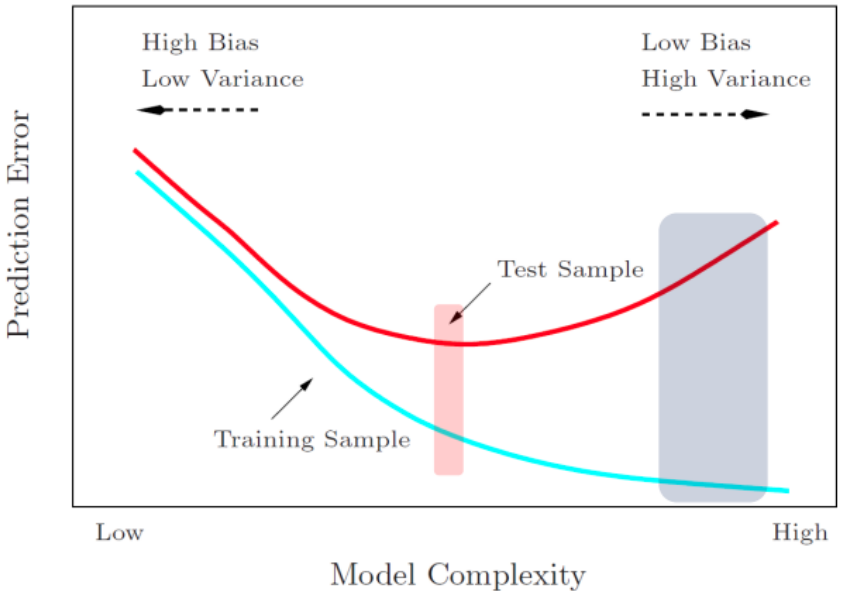
1-Nearest Neighbor Classifier

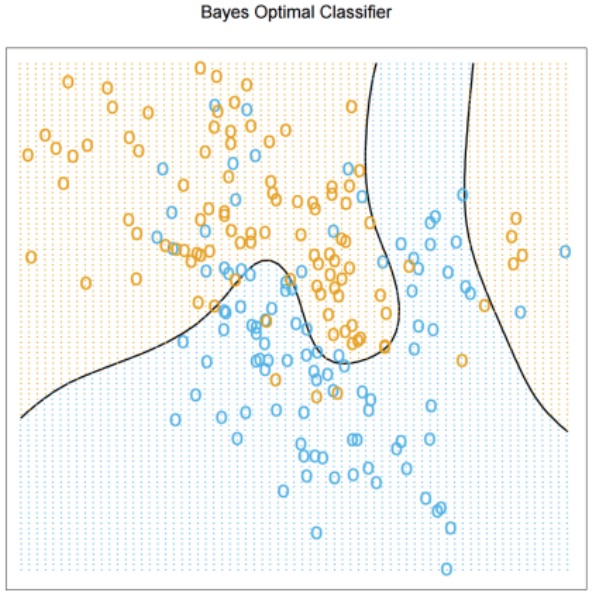


Complex models have low bias, but high variance.

Bias / Variance Trade-off

We want to reduce both sources of error to obtain a model that is 'just right'.





However, finding the right complexity of the model is not a simple matter.

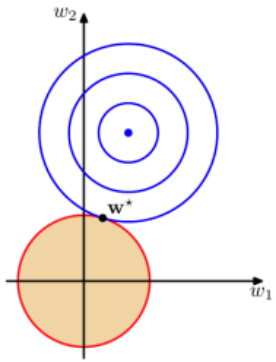
Most regularization methods aim to reduce variance at the cost of added bias.

Previous class: L2- vs L1-regularization

L2 regularization promotes **grouping** – results in equal weights for correlated features

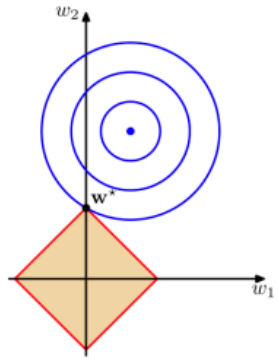
L1 regularization promotes **sparsity** – selects few informative features

L2-regularization



$$\Omega(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$$

L1-regularization



$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1$$

Regularization: Expressing Preferences

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

$$x = [1, 1, 1, 1]$$
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_1^T x = w_2^T x = 1$$

Same predictions, so data loss will always be the same

Regularization: Expressing Preferences

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$\boxed{w_2 = [0.25, 0.25, 0.25, 0.25]}$$

L2 Regularization

$$R(W) = \sum_{k,l} W_{k.l}^2$$

L2 regularization prefers weights to be "spread out"

$$w_1^T x = w_2^T x = 1$$

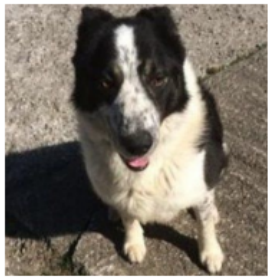
Same predictions, so data loss will always be the same

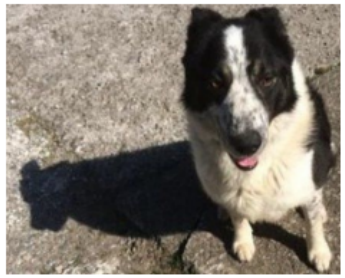
Algorithm	Tracks first moments (Momentum)	Tracks second moments (Adaptive learning rates)	Leaky second moments	Bias correction for moment estimates
SGD	✗	✗	✗	✗
SGD+Momentum	✓	✗	✗	✗
Nesterov	✓	✗	✗	✗
AdaGrad	✗	✓	✗	✗
RMSProp	✗	✓	✓	✗
Adam	✓	✓	✓	✓

Data augmentation

The best way to make a machine learning model generalize better is to train it with more data – however, in practice, the amount of data we have is limited.





Then, we can artificially enlarge the data by generating some variation of inputs e.g.) Flipping, cropping, translating, rotating, and etc, for image inputs

Data augmentation: More data is better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- Get around the problem by creating synthesized data
- For some ML tasks it is straightforward to synthesize data

Data augmentation: More data is better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- Get around the problem by creating synthesized data
- For some ML tasks it is straightforward to synthesize data

Data augmentation: More data is better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- Get around the problem by creating synthesized data
- For some ML tasks it is straightforward to synthesize data

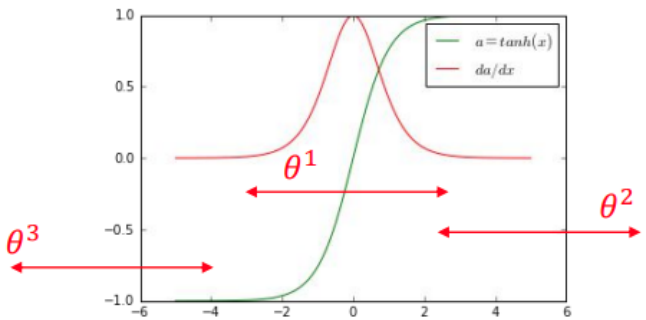
Data augmentation: More data is better

- Best way to make a ML model to generalize better is to train it on more data
- In practice amount of data is limited
- Get around the problem by creating synthesized data
- For some ML tasks it is straightforward to synthesize data

Data preprocessing

- Scale input variables to have similar diagonal covariances
- Similar covariances $\rightarrow$ more balanced rate of learning for different weights
- Rescaling to 1 is a good choice, unless some dimensions are less important

$x = [x^1, x^2, x^3]^T, \theta = [\theta^1, \theta^2, \theta^3]^T, a = \tanh(\theta^T x)$



$x^1, x^2, x^3 \rightarrow$ much different covariances

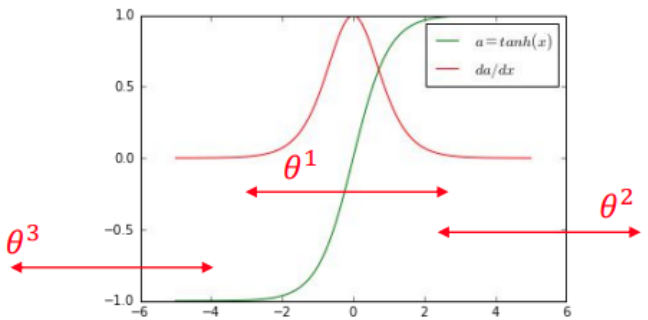
Generated gradients $\left.\frac{\partial \mathcal{L}}{\partial \theta}\right|_{x^1, x^2, x^3}$: much different

Gradient update harder: $\theta^{t+1} = \theta^t - \eta_t \begin{bmatrix} \partial \mathcal{L}/\partial \theta^1 \\ \partial \mathcal{L}/\partial \theta^2 \\ \partial \mathcal{L}/\partial \theta^3 \end{bmatrix}$

Data preprocessing

- Scale input variables to have similar diagonal covariances
- Similar covariances $\rightarrow$ more balanced rate of learning for different weights
- Rescaling to 1 is a good choice, unless some dimensions are less important

$x = [x^1, x^2, x^3]^T, \theta = [\theta^1, \theta^2, \theta^3]^T, a = \tanh(\theta^T x)$



$x^1, x^2, x^3 \rightarrow$ much different covariances

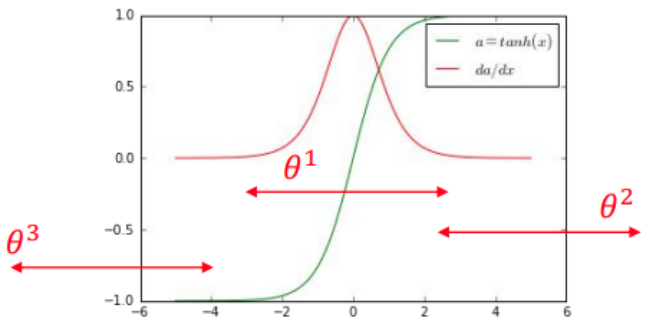
Generated gradients $\left.\frac{\partial \mathcal{L}}{\partial \theta}\right|_{x^1, x^2, x^3}$: much different

Gradient update harder: $\theta^{t+1} = \theta^t - \eta_t \begin{bmatrix} \partial \mathcal{L}/\partial \theta^1 \\ \partial \mathcal{L}/\partial \theta^2 \\ \partial \mathcal{L}/\partial \theta^3 \end{bmatrix}$

Data preprocessing

- Scale input variables to have similar diagonal covariances
- Similar covariances $\rightarrow$ more balanced rate of learning for different weights
- Rescaling to 1 is a good choice, unless some dimensions are less important

$x = [x^1, x^2, x^3]^T, \theta = [\theta^1, \theta^2, \theta^3]^T, a = \tanh(\theta^T x)$



$x^1, x^2, x^3 \rightarrow$ much different covariances

Generated gradients $\left.\frac{\partial \mathcal{L}}{\partial \theta}\right|_{x^1, x^2, x^3}$: much different

Gradient update harder: $\theta^{t+1} = \theta^t - \eta_t \begin{bmatrix} \partial \mathcal{L}/\partial \theta^1 \\ \partial \mathcal{L}/\partial \theta^2 \\ \partial \mathcal{L}/\partial \theta^3 \end{bmatrix}$

Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

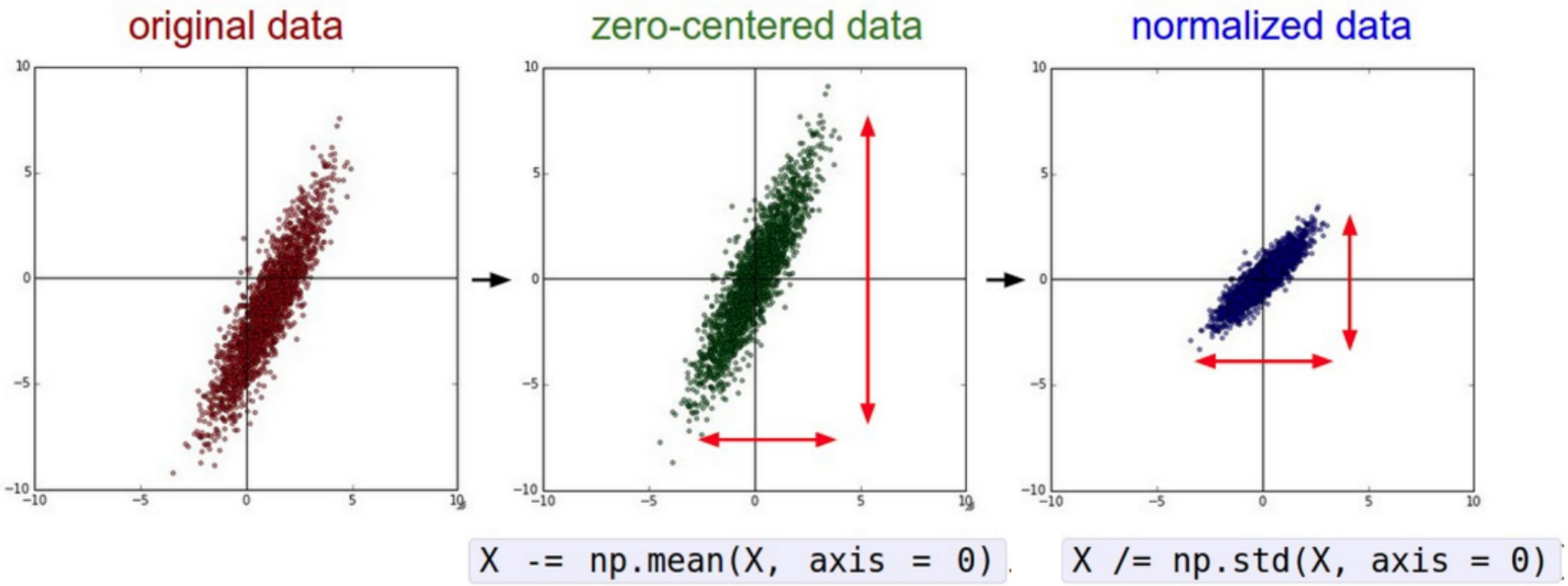
Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

Data preprocessing

- Input variables should be as decorrelated as possible
  - Input variables are "more independent"
  - Network is forced to find non-trivial correlations between inputs
  - Decorrelated inputs $\rightarrow$ Better optimization
  - Obviously not the case when inputs are by definition correlated (sequences)
  - Extreme case: Extreme correlation (linear dependency) might cause problems

Data preprocessing



Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout

- Best way to regularize a fixed size model is
  - Average the predictions of all possible settings of the parameters
  - Weighting each setting with the posterior probability given the training data
    - This would be the Bayesian approach
- Dropout does this using considerably less computation
  - By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters

Dropout is a bagging method

- Bagging is a method of averaging over several models to improve generalization
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks
    - It is a method of bagging applied to neural networks
- Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout is a bagging method

- Bagging is a method of averaging over several models to improve generalization
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks
    - It is a method of bagging applied to neural networks
- Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout is a bagging method

- Bagging is a method of averaging over several models to improve generalization
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks
    - It is a method of bagging applied to neural networks
- Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout is a bagging method

- Bagging is a method of averaging over several models to improve generalization
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks
    - It is a method of bagging applied to neural networks
- Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout is a bagging method

- Bagging is a method of averaging over several models to improve generalization
- Impractical to train many neural networks since it is expensive in time and memory
  - Dropout makes it practical to apply bagging to very many large neural networks
    - It is a method of bagging applied to neural networks
- Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout: Removing units creates networks

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

Dropout: Removing units creates networks

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

Dropout: Removing units creates networks

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

Dropout: Removing units creates networks

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

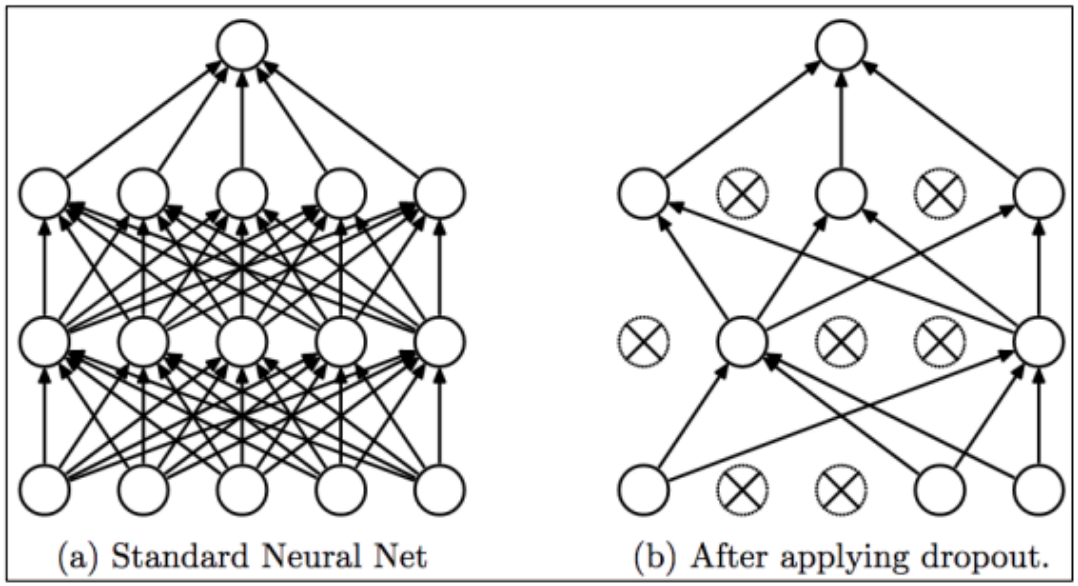
Dropout: Removing units creates networks

- Dropout trains an ensemble of all subnetworks
  - Subnetworks formed by removing non-output units from an underlying base network
- We can effectively remove units by multiplying its output value by zero
  - For networks based on performing a series of affine transformations or on-linearities
  - Needs some modification for radial basis functions based on difference between unit state and a reference value

Dropout Neural Net

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.

- A simple way to prevent neural net overfitting



Drop hidden and visible units from net, i.e., temporarily remove it from the network with all input/output connections. Choice of units to drop is random, determined by a probability p, chosen by a validation set, or equal to 0.5

(a) A standard neural net with two hidden layers

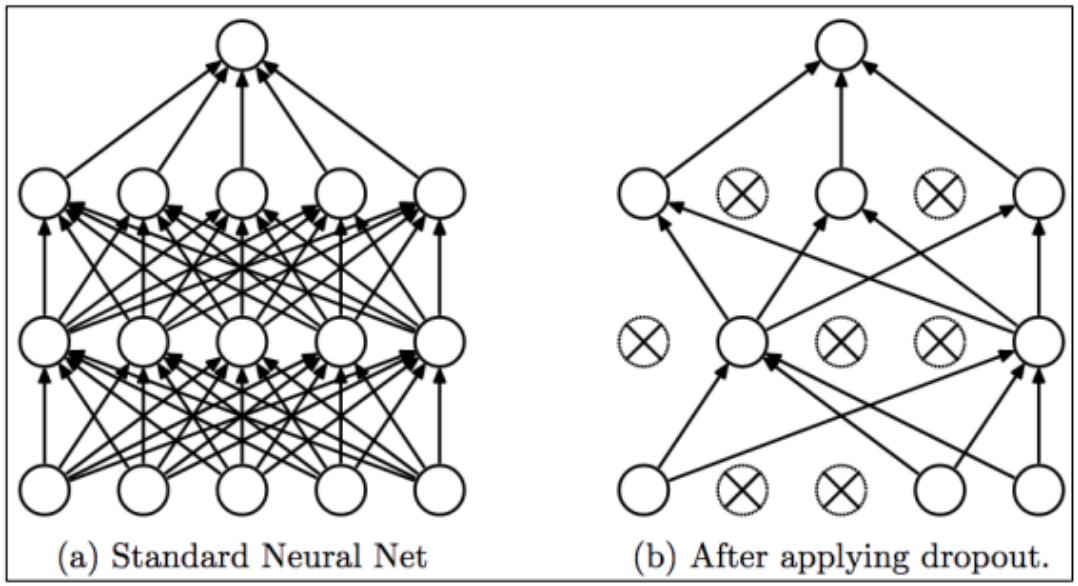
(b) A thinned net produced by applying dropout, crossed units have been dropped

Bernoulli Distribution

Dropout Neural Net

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.

- A simple way to prevent neural net overfitting



Drop hidden and visible units from net, i.e., temporarily remove it from the network with all input/output connections. Choice of units to drop is random, determined by a probability p, chosen by a validation set, or equal to 0.5

(a) A standard neural net with two hidden layers

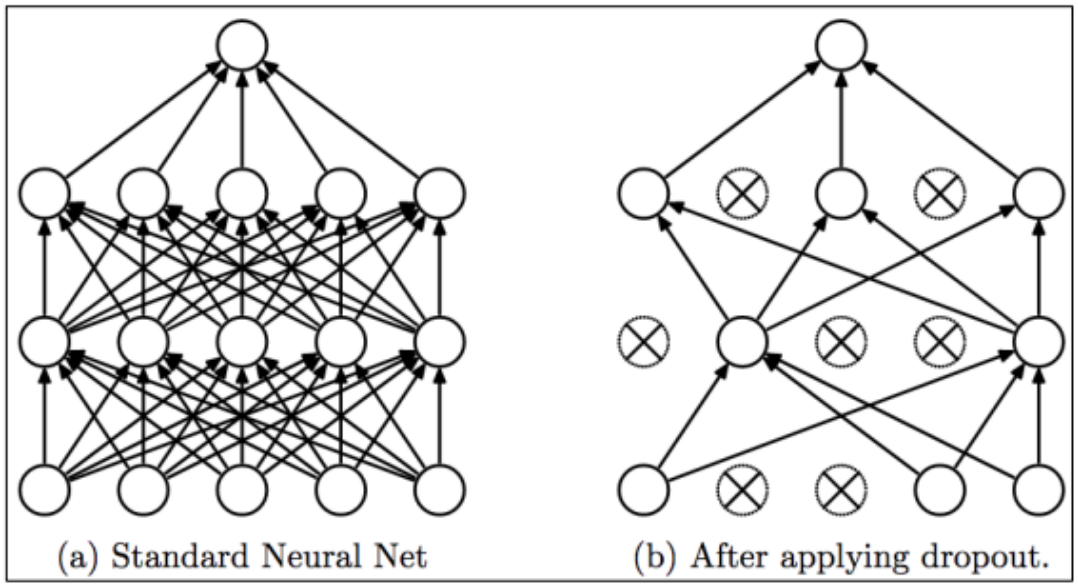
(b) A thinned net produced by applying dropout, crossed units have been dropped

Bernoulli Distribution

Dropout Neural Net

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.

- A simple way to prevent neural net overfitting



Drop hidden and visible units from net, i.e., temporarily remove it from the network with all input/output connections. Choice of units to drop is random, determined by a probability p, chosen by a validation set, or equal to 0.5

(a) A standard neural net with two hidden layers

(b) A thinned net produced by applying dropout, crossed units have been dropped

Bernoulli Distribution

Dropout as bagging

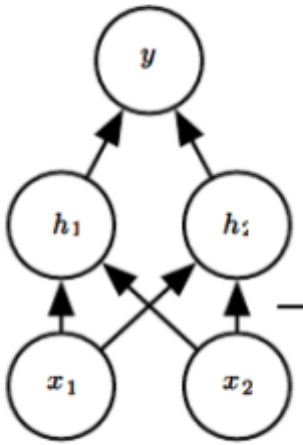
- In bagging we define k different models, construct k different data sets by sampling from the dataset with replacement, and train model i on dataset i
- Dropout aims to approximate this process, but with an exponentially large no. of neural networks

Dropout as bagging

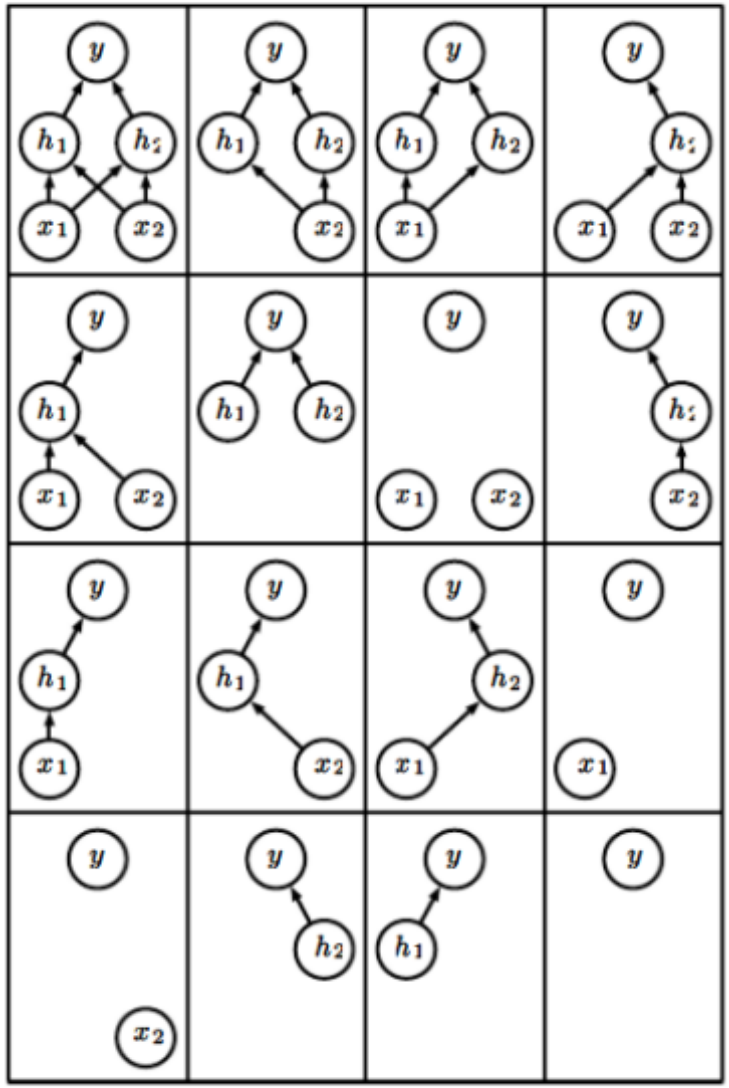
- In bagging we define k different models, construct k different data sets by sampling from the dataset with replacement, and train model i on dataset i
- Dropout aims to approximate this process, but with an exponentially large no. of neural networks

Dropout as an ensemble method

- Remove non-output units from base network.
- Remaining 4 units yield 16 networks



Base network


Ensemble of subnetworks

- Here many networks have no path from input to output
- Problem insignificant with large networks

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter setting

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of subnetworks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of sub-networks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of sub-networks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

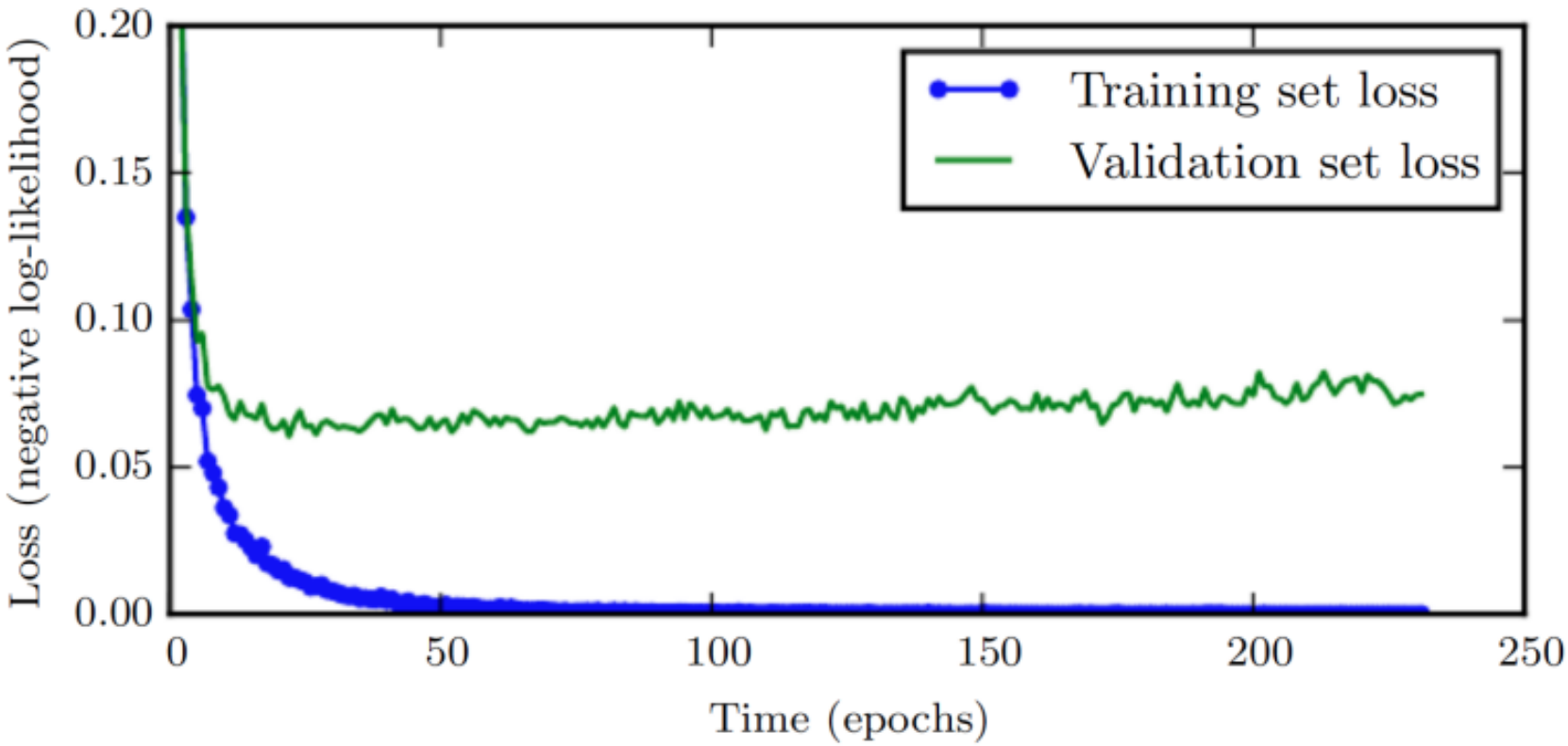
- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of sub-networks are trained for a single step
    - Parameter sharing allows good parameter settings

Bagging training vs Dropout training

- Dropout training not same as bagging training
  - In bagging, the models are all independent
  - In dropout, models share parameters
    - Models inherit subsets of parameters from parent network
    - Parameter sharing allows an exponential no. of models with a tractable amount of memory
- In bagging each model is trained to convergence on its respective training set
  - In dropout, most models are not explicitly trained
    - Fraction of sub-networks are trained for a single step
    - Parameter sharing allows good parameter settings

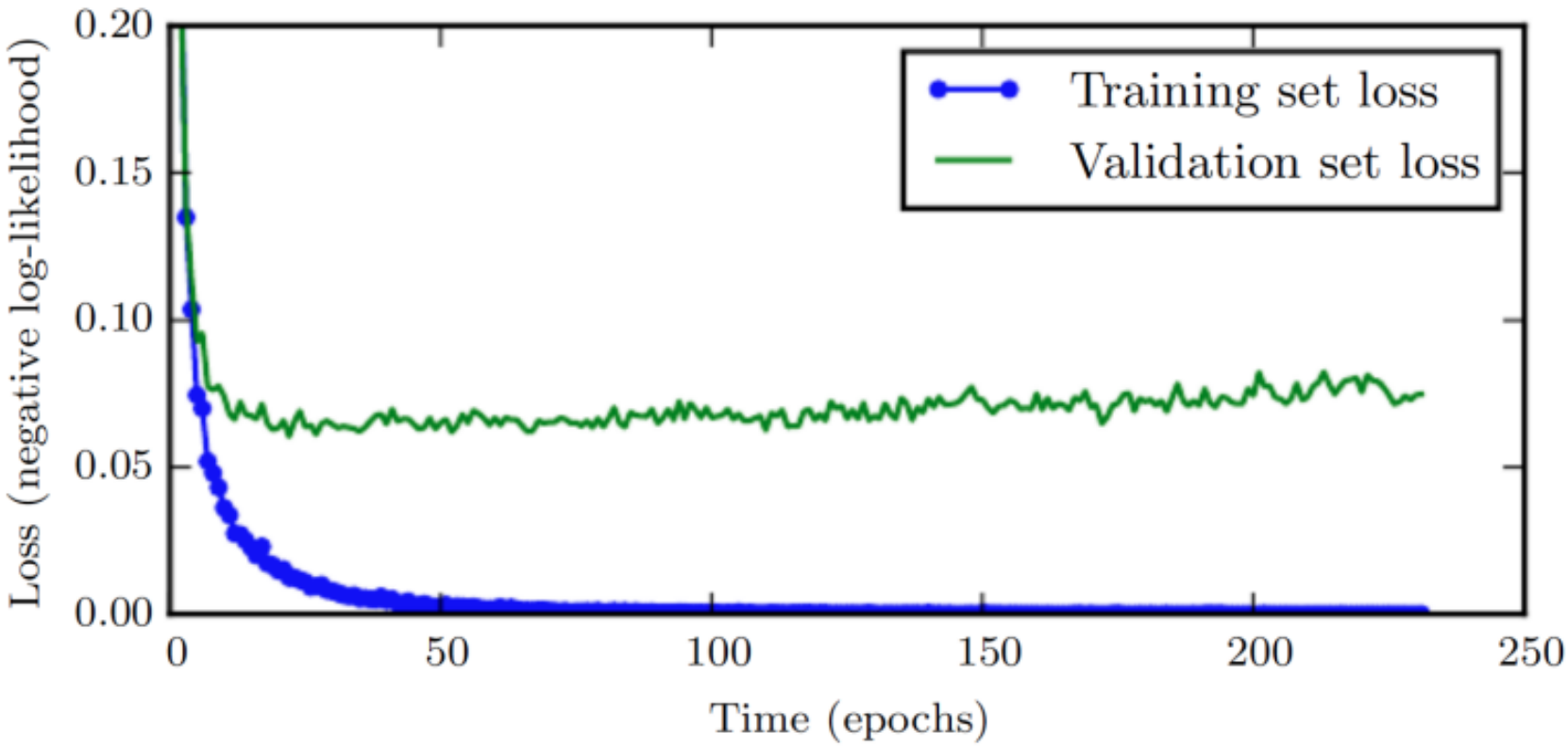
Early stopping:Increase in validation set error

- When training large models with sufficient representational capacity to overfit the task, training error decreases steadily over time, but validation set error begins to rise again
- An example of this behavior is shown next



Early stopping:Increase in validation set error

- When training large models with sufficient representational capacity to overfit the task, training error decreases steadily over time, but validation set error begins to rise again
- An example of this behavior is shown next



Early stopping

- We can thus obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set

Early stopping

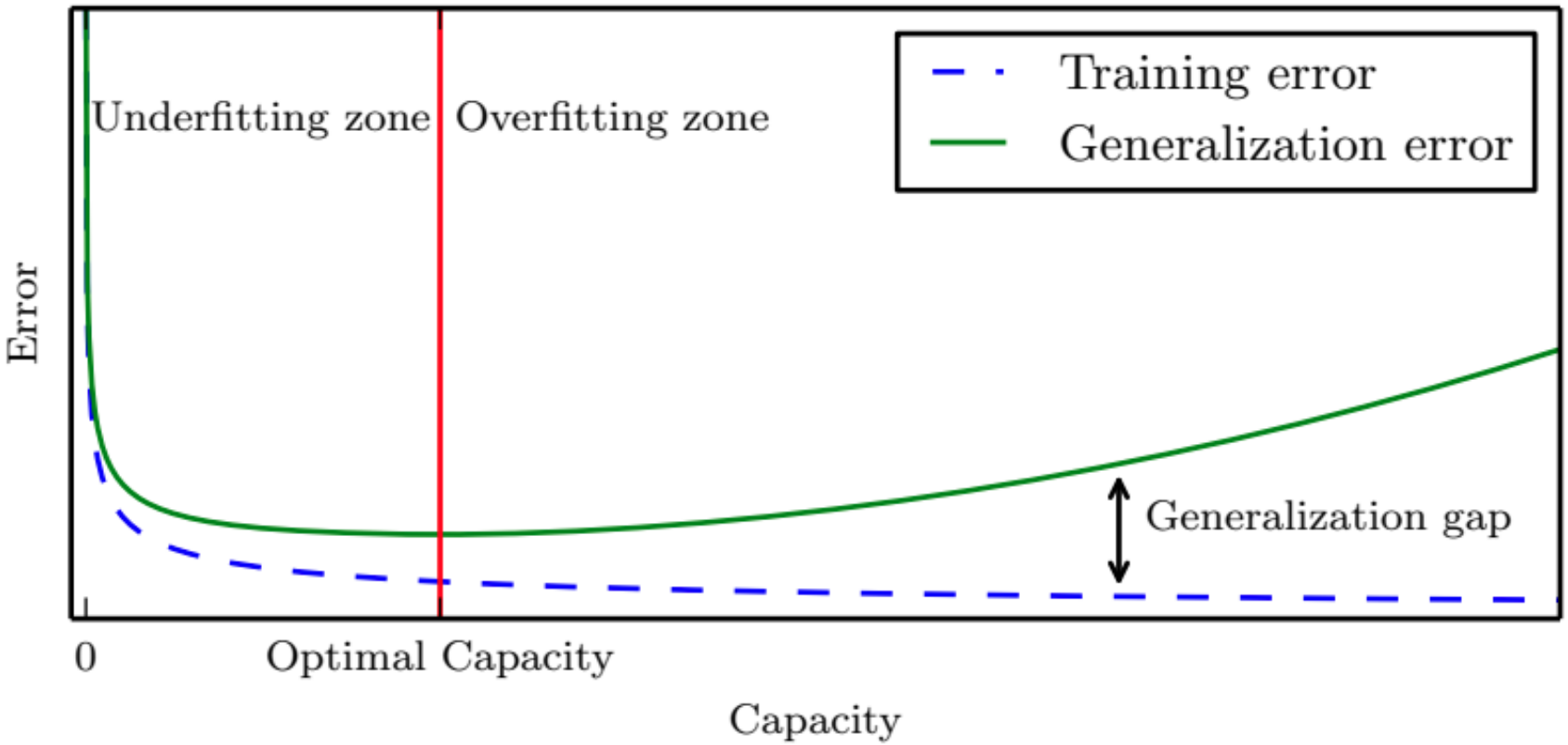
- We can thus obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set

Early stopping

- We can thus obtain a model with better validation set error (and thus better test error) by returning to the parameter setting at the point of time with the lowest validation set error
- Every time the error on the validation set improves, we store a copy of the model parameters.
- When the training algorithm terminates, we return these parameters, rather than the latest set

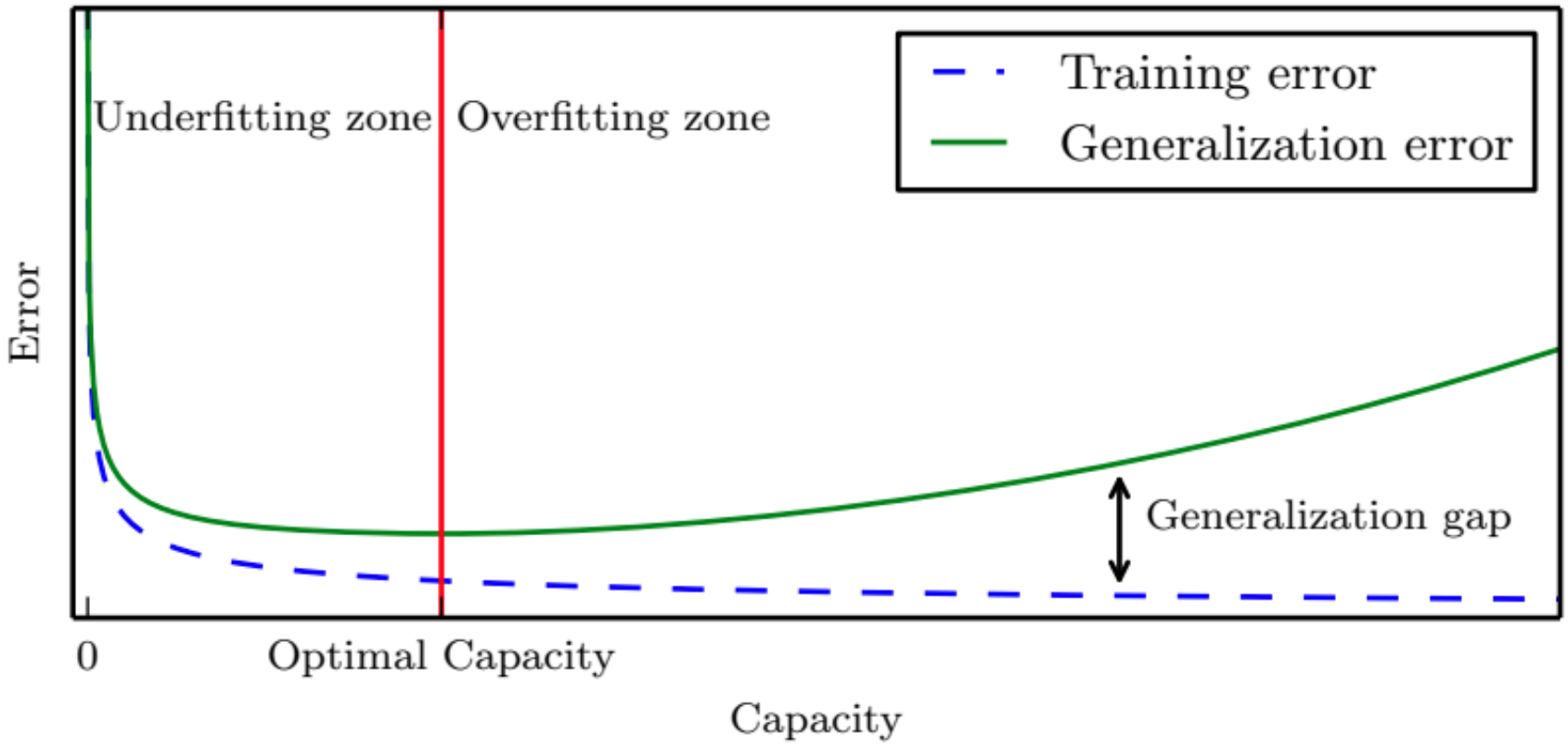
Early stopping

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.



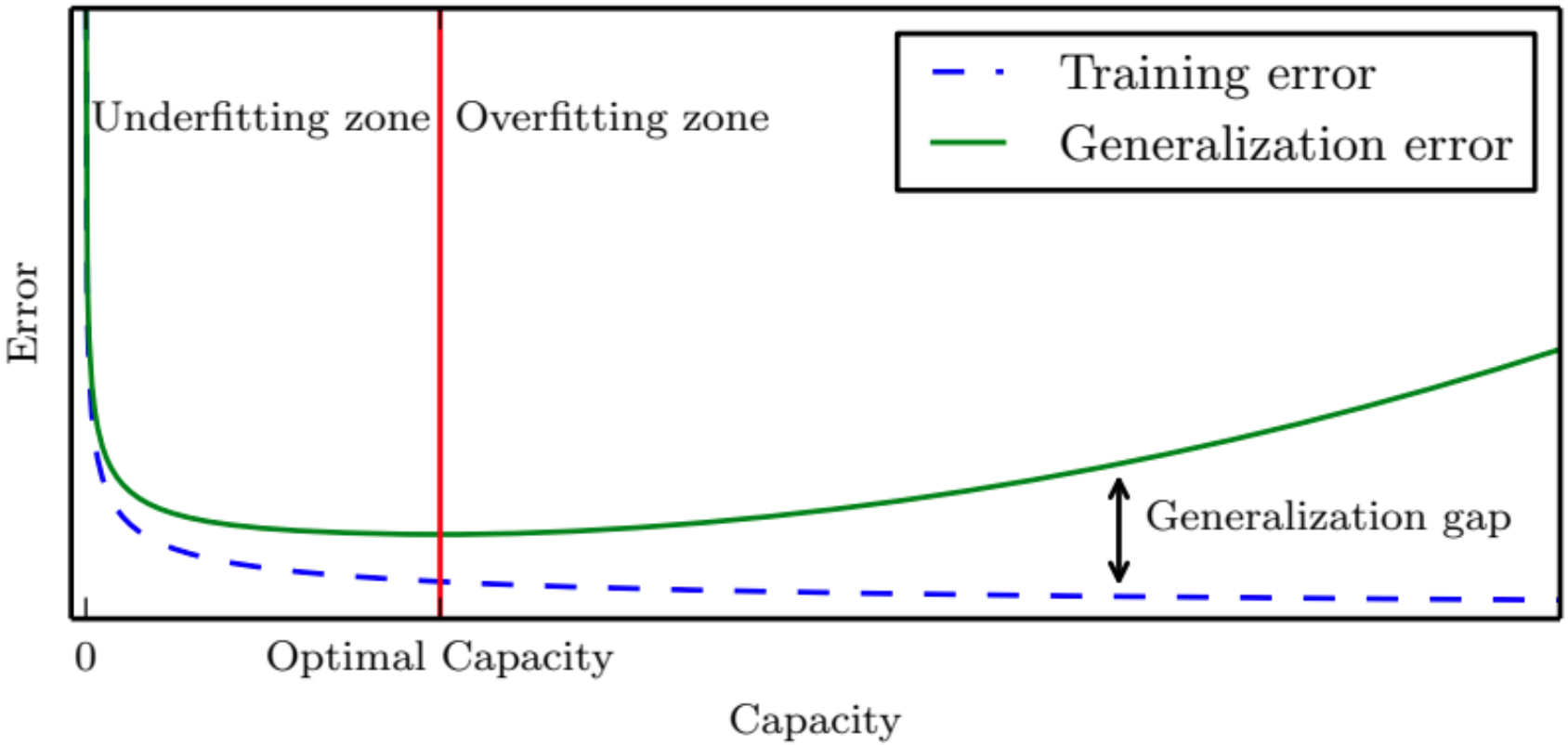
Early stopping

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.



Early stopping

- Start with small weights and stop the learning before it overfits.
- Think early stopping as a very efficient hyperparameter selection.
- The number of training steps is just another hyperparameter.



Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed
  - In the second extra training step, all the training data is included
- There are two basic strategies for the second training procedure

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed
  - In the second extra training step, all the training data is included
- There are two basic strategies for the second training procedure

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed
  - In the second extra training step, all the training data is included
- There are two basic strategies for the second training procedure

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed
  - In the second extra training step, all the training data is included
- There are two basic strategies for the second training procedure

Use of a second training step

- Early stopping requires a validation set
  - Thus some training data is not fed to the model
- To best exploit this extra data, one can perform extra training after the initial training with early stopping has completed
  - In the second extra training step, all the training data is included
- There are two basic strategies for the second training procedure

First Strategy for Retraining

- One strategy is to initialize the model again and retrain on all the data
- In the second training pass, we train for the same no. of steps as the early stopping procedure determined was optimal in first pass
- Whether to retrain for the same no. of parameter updates or the same no of passes through the data set?
  - On the second round, each pass through dataset will require more parameter updates because dataset is bigger

First Strategy for Retraining

- One strategy is to initialize the model again and retrain on all the data
- In the second training pass, we train for the same no. of steps as the early stopping procedure determined was optimal in first pass
- Whether to retrain for the same no. of parameter updates or the same no of passes through the data set?
  - On the second round, each pass through dataset will require more parameter updates because dataset is bigger

First Strategy for Retraining

- One strategy is to initialize the model again and retrain on all the data
- In the second training pass, we train for the same no. of steps as the early stopping procedure determined was optimal in first pass
- Whether to retrain for the same no. of parameter updates or the same no of passes through the data set?
    - On the second round, each pass through dataset will require more parameter updates because dataset is bigger

First Strategy for Retraining

- One strategy is to initialize the model again and retrain on all the data
- In the second training pass, we train for the same no. of steps as the early stopping procedure determined was optimal in first pass
- Whether to retrain for the same no. of parameter updates or the same no of passes through the data set?
  - On the second round, each pass through dataset will require more parameter updates because dataset is bigger

Second strategy for retraining

- Keep all the parameters obtained from the first round of training and then continue training but now require using all the data
- We no longer have a guide for when to stop in terms of the no of steps
- Instead we monitor the average loss function on the validation set and continue training until it falls below the value of the training set objective of when early stopping halted

Second strategy for retraining

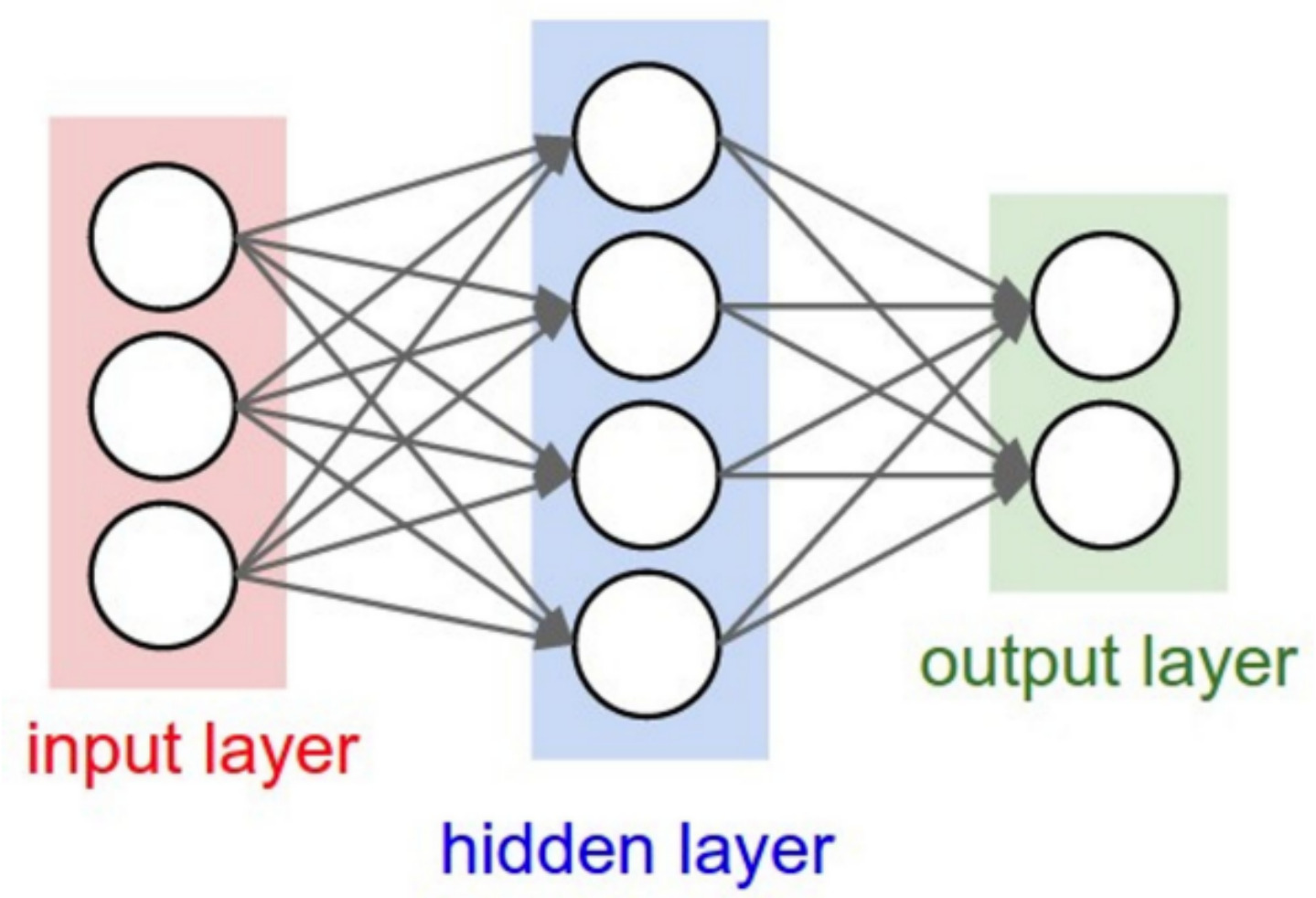
- Keep all the parameters obtained from the first round of training and then continue training but now require using all the data
- We no longer have a guide for when to stop in terms of the no of steps
- Instead we monitor the average loss function on the validation set and continue training until it falls below the value of the training set objective of when early stopping halted

Second strategy for retraining

- Keep all the parameters obtained from the first round of training and then continue training but now require using all the data
- We no longer have a guide for when to stop in terms of the no of steps
- Instead we monitor the average loss function on the validation set and continue training until it falls below the value of the training set objective of when early stopping halted

Weight Initialization

Q: what happens when W=0 init is used?



Weight Initialization

First idea: Small random numbers (Gaussian with zero mean and 1e-2 standard deviation)

$$w = 0.001 * np.random.rand(D, H)$$

- Works okay for small networks, but can lead to non-homogeneous distributions of activations across the layers of a network.

Weight Initialization

Lets look at some activation statistics

E.g. 10-layer net with 500 neurons on each layer, using tanh non-linearities, and initializing as described in last slide.

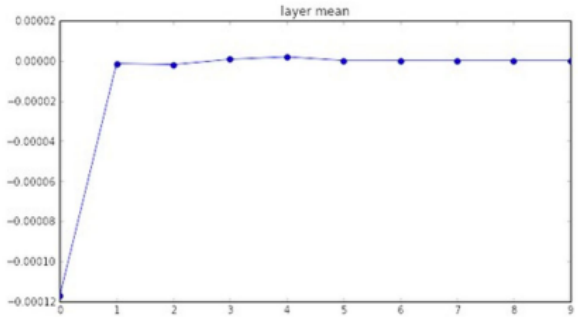
```
# assume some unit gaussian 10-D input data
D = np.random.randn(1000, 500)
hidden_layer_sizes = [500]*10
nonlinearities = ['tanh']*len(hidden_layer_sizes)
```

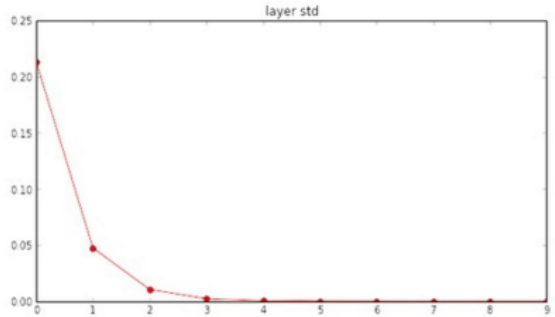
```
act = {'relu':lambda x:np.maximum(0,x), 'tanh':lambda x:np.tanh(x)}
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i == 0 else Hs[i-1] # input at this layer
    fan_in = X.shape[1]
    fan_out = hidden_layer_sizes[i]
    W = np.random.randn(fan_in, fan_out) * 0.01 # layer initialization

    H = np.dot(X, W) # matrix multiply
    H = act[nonlinearities[i]](H) # nonlinearity
    Hs[i] = H # cache result on this layer
```

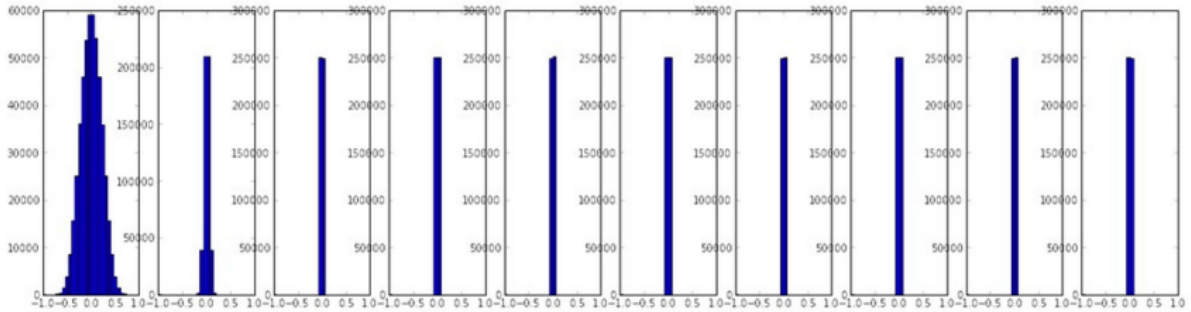
```
# look at distributions at each layer
print 'input layer had mean %f and std %f' % (np.mean(D), np.std(D))
layer_means = [np.mean(H) for i,H in Hs.iteritems()]
layer_stds = [np.std(H) for i,H in Hs.iteritems()]
for i,H in Hs.iteritems():
    print 'hidden layer %d had mean %f and std %f' % (i+1, layer_means[i], layer_stds[i])

# plot the means and standard deviations
plt.figure()
plt.subplot(121)
plt.plot(Hs.keys(), layer_means, 'ob-')
plt.title('layer mean')
plt.subplot(122)
plt.plot(Hs.keys(), layer_stds, 'or-')
plt.title('layer std')

# plot the raw distributions
plt.figure()
for i,H in Hs.iteritems():
    plt.subplot(1,len(Hs),i+1)
    plt.hist(H.ravel(), 30, range=(-1,1))
```

Weight Initialization

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```







All activations become zero!

Q: think about the backward pass. What do the gradients look like?

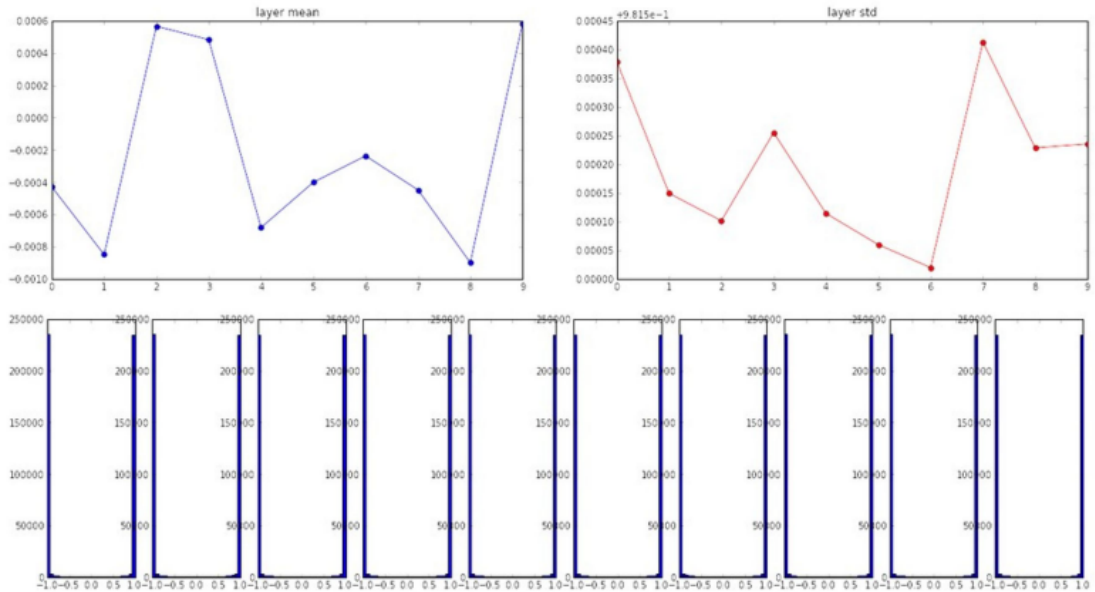
Hint: think about backward pass for a W*X gate.

Weight Initialization

```
W = np.random.randn(fan_in, fan_out) * 1.0 # layer initialization
```

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

*1.0 instead of *0.01



Almost all neurons completely saturated, either -1 and 1. Gradients will be all zero.

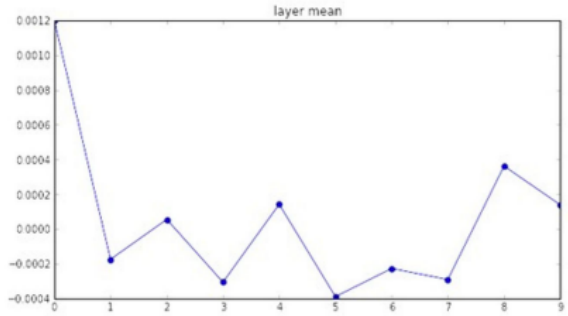
Weight Initialization

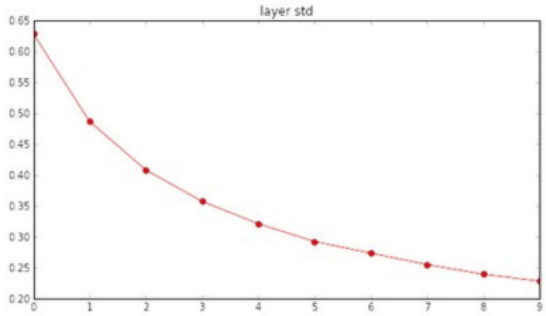
```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

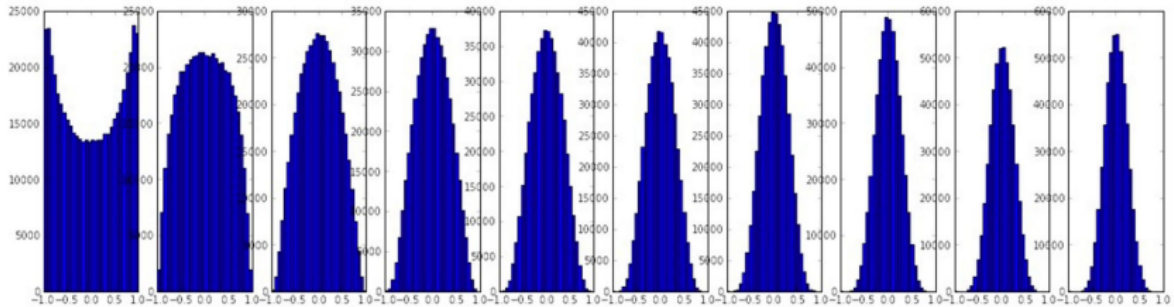
```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

Keep the variance the same across every layer!

"Xavier initialization" [Glorot et al., 2010]







Reasonable initialization. (Mathematical derivation assumes linear activations)

- If a hidden unit has a big fan-in, small changes on many of its incoming weights can cause the learning to overshoot.
  - We generally want smaller incoming weights when the fan-in is big, so initialize the weights to be proportional to sqrt(fan-in).
- We can also scale the learning rate the same way. More on this later!

(from Hinton's notes)

Weight Initialization

Proper initialization is an active area of research

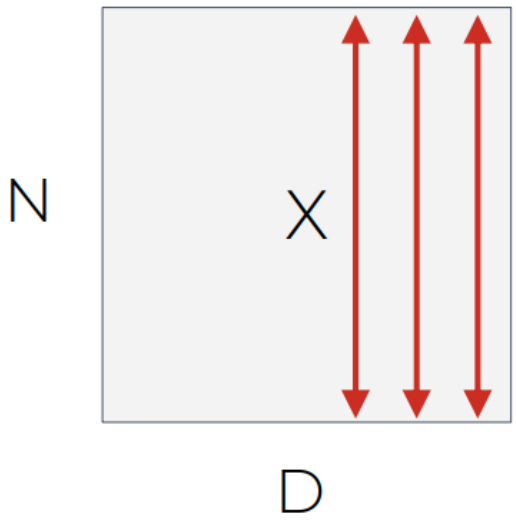
- Understanding the difficulty of training deep feedforward neural networks. Glorot and Bengio, 2010
- Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. Saxe et al, 2013
- Random walk initialization for training very deep feedforward networks. Sussillo and Abbott, 2014
- Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. He et al., 2015
- Data-dependent Initializations of Convolutional Neural Networks. Krähenbühl et al., 2015
- All you need is a good init. Mishkin and Matas, 2015
- How to start training: The effect of initialization and architecture. Hanin and Rolnick, 2018
- How to Initialize your Network? Robust Initialization for WeightNorm

Batch Normalization

consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}x^{(k)}}{\sqrt{Var[x^{(k)}]}}$$

Batch Normalization

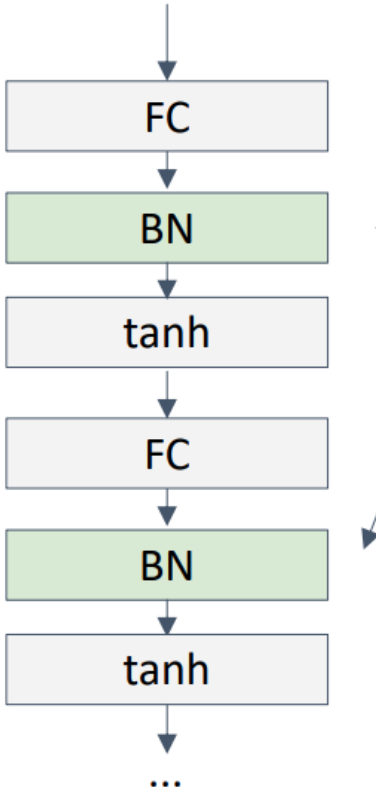


1. compute the empirical mean and variance independently for each dimension.

2. Normalize

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization



Usually inserted after Fully Connected / (or Convolutional, as we'll see soon) layers, and before nonlinearity.

Problem: do we necessarily want a unit Gaussian input to a tanh layer?

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

[Ioffe and Szegedy, 2015]

Batch Normalization

Normalize:

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

And then allow the network to squash the range if it wants to:

$$y^{(k)} = \gamma^{(k)} \widehat{x}^{(k)} + \beta^{(k)}$$

Note, the network can learn:

$$\gamma^{(k)} = \sqrt{\mathrm{Var}[x^{(k)}]}$$

$$\beta^{(k)} = \mathrm{E}[x^{(k)}]$$

to recover the identity mapping.

[Ioffe and Szegedy, 2015]

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

[Ioffe and Szegedy, 2015]

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)

[Ioffe and Szegedy, 2015]

Hyperparameter Selection

Everything is a hyperparameter

- Network size/depth
- Small model variations
- Minibatch creation strategy
- Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!

Hyperparameter Selection

Everything is a hyperparameter

- Network size/depth
- Small model variations
- Minibatch creation strategy
- Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!

Hyperparameter Selection

Everything is a hyperparameter

- Network size/depth
- Small model variations
- Minibatch creation strategy
- Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!

Hyperparameter Selection

Everything is a hyperparameter

- Network size/depth
- Small model variations
- Minibatch creation strategy
- Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!

Hyperparameter Selection

Everything is a hyperparameter

- Network size/depth
- Small model variations
- Minibatch creation strategy
- Optimizer/learning rate
- Models are complicated and opaque, debugging can be difficult!

Cross-validation strategy

First do coarse -> fine cross-validation in stages

- First stage: only a few epochs to get rough idea of what params work
- Second stage: longer running time, finer search
- ....(repeat as necessary)

Cross-validation strategy

First do coarse -> fine cross-validation in stages

- First stage: only a few epochs to get rough idea of what params work
- Second stage: longer running time, finer search
- ....(repeat as necessary)

Cross-validation strategy

First do coarse -> fine cross-validation in stages

- First stage: only a few epochs to get rough idea of what params work
- Second stage: longer running time, finer search
- ....(repeat as necessary)

Hyperparameters to play with

- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

Hyperparameters to play with

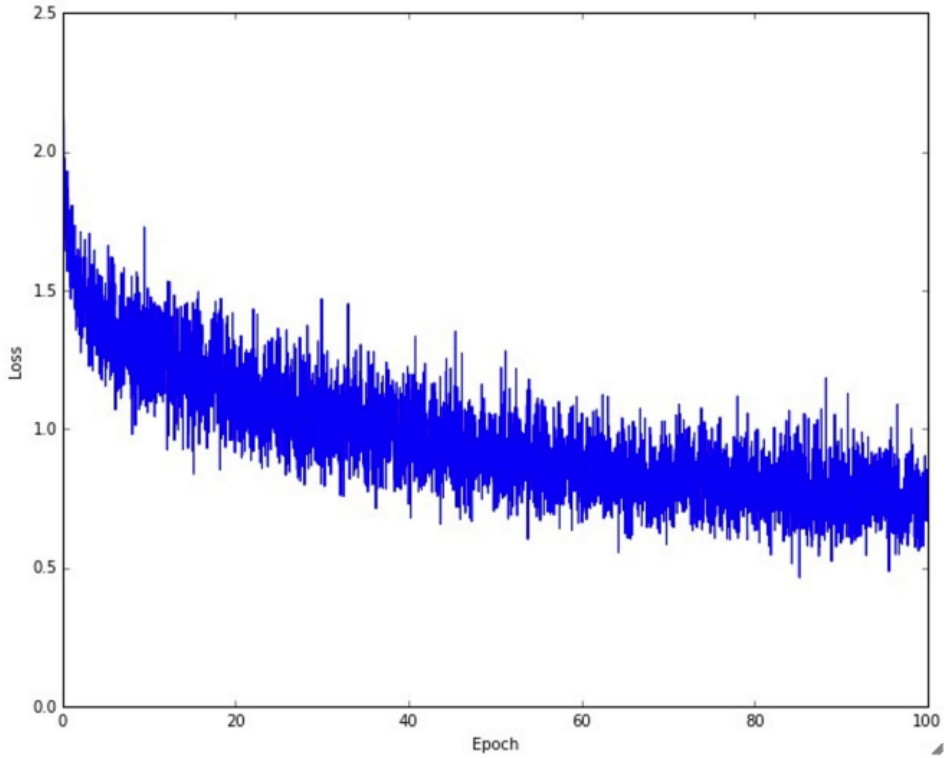
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

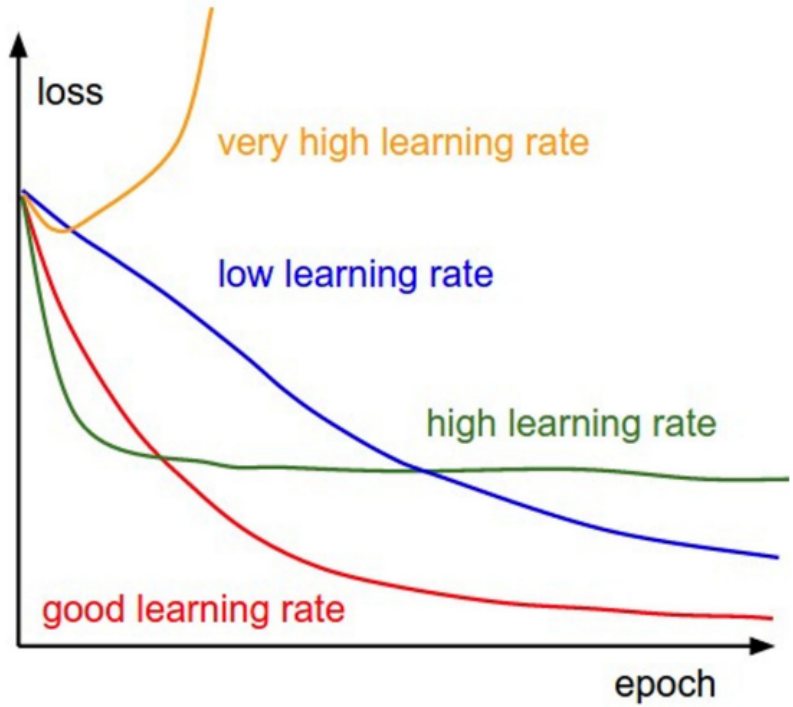
Hyperparameters to play with

- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

Monitor and visualize the loss curve

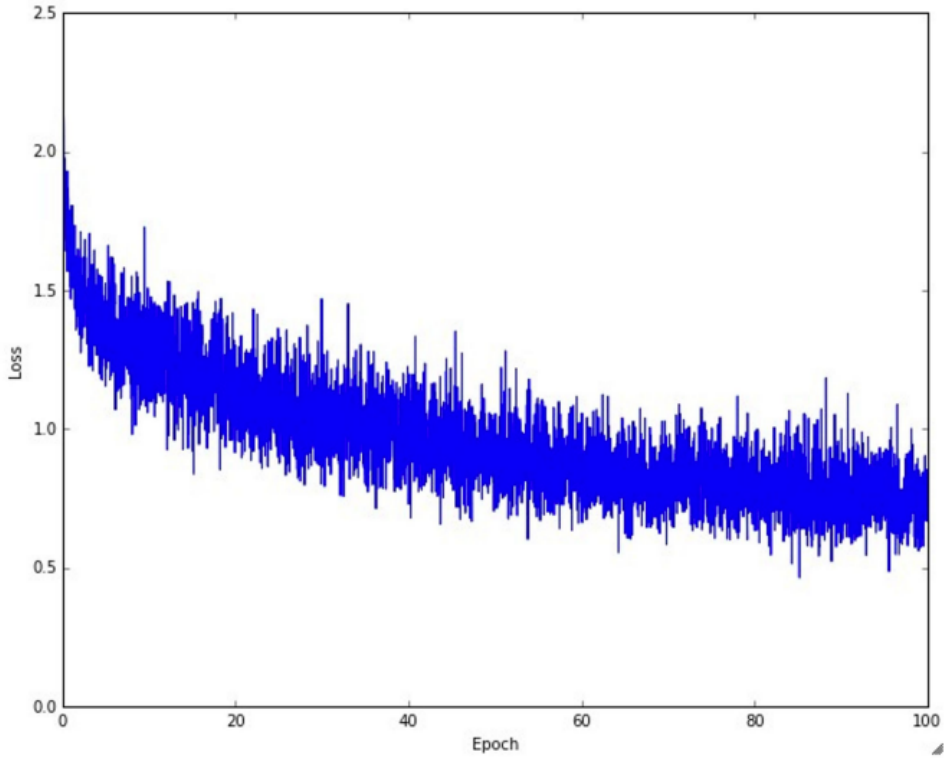
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

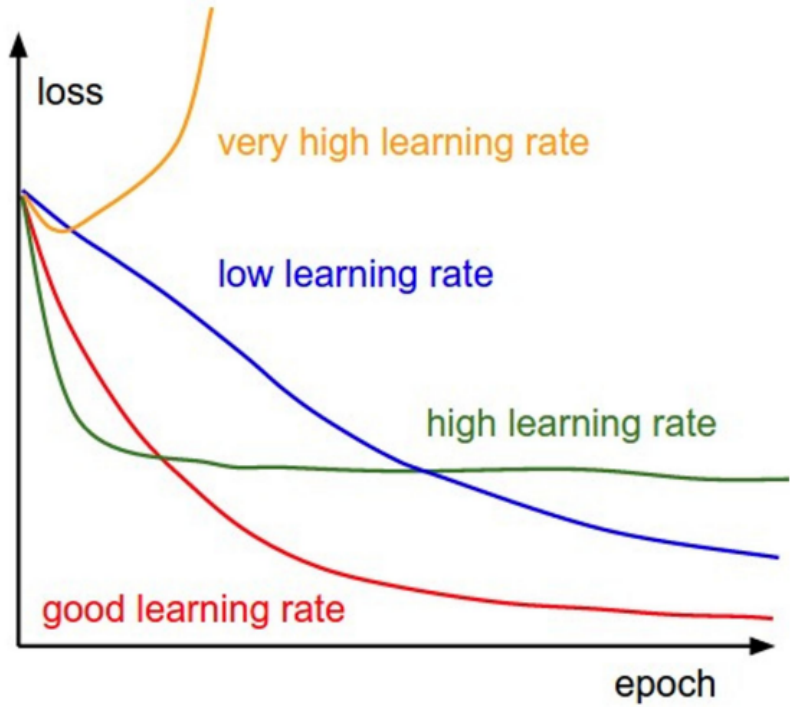




Monitor and visualize the loss curve

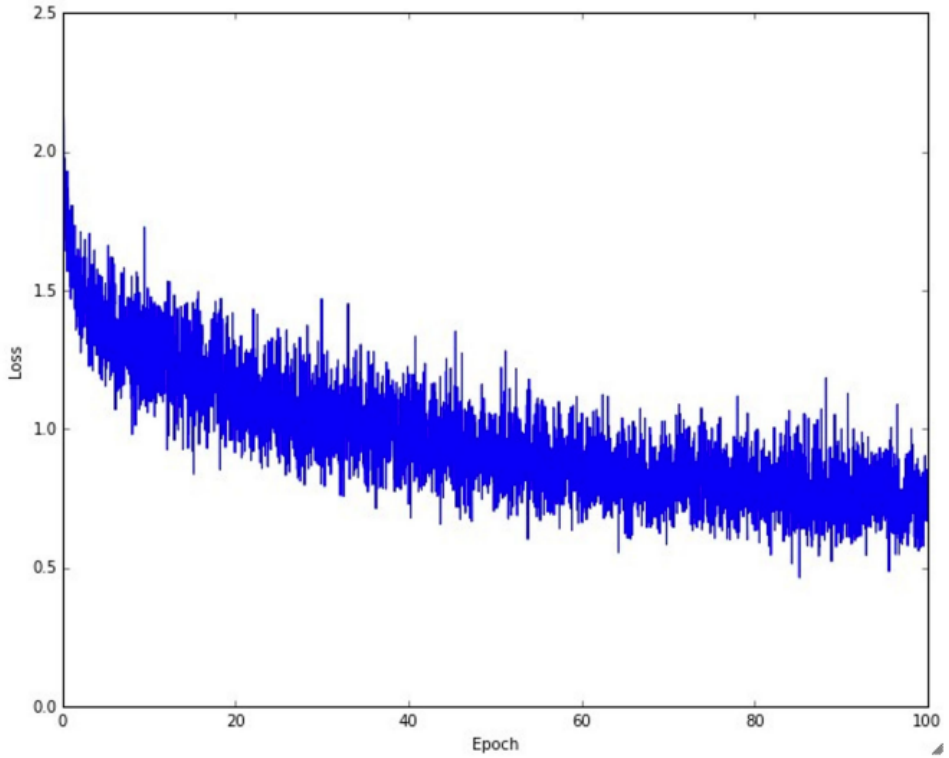
- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)

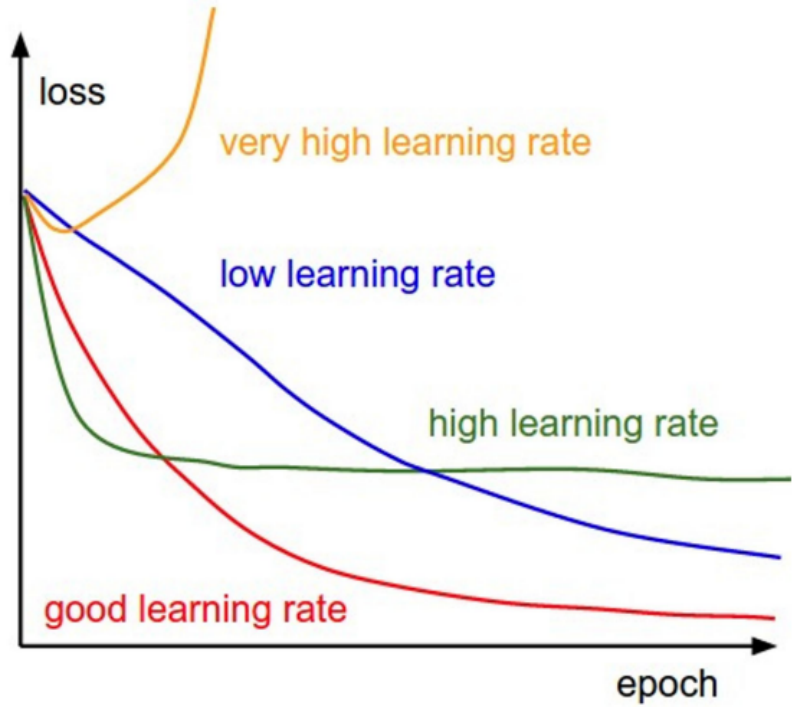




Monitor and visualize the loss curve

- network architecture
- learning rate, its decay schedule, update type
- regularization (L2/Dropout strength)





CNN

Next Class..